READ-OVERFLOW

Must ensure that the buffer is large enough to hold the number of bytes read

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-02

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 5896 bytes

Attack Category	• Malicious Input
Vulnerability Category	• Buffer Overflow • Input source (not really attack)
Software Context	• File I/O
Location	• unistd.h
Description	The read function attempts to read nbyte bytes from the file associated with the open file descriptor, fildes, into the buffer pointed to by buf. If nbyte is 0, read will return 0 and have no other results. On files that support seeking (for example, a regular file), the read starts at a position in the file given by the file offset associated with fildes. The file offset is incremented by the number of bytes actually read. Files that do not support seeking (for example, terminals) always read from the current position. The value of a file offset associated with such a file is undefined. If fildes refers to a socket, read is equivalent to recv (3SOCKET) with no flags set. No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent read requests is implementation dependent. If the value of nbyte is greater than SSIZE_MAX, the result is implementation dependent. The developer must ensure that the buffer is large enough to hold the number of bytes read. This is most commonly a problem when an input file stream contains a 'count' for the number of bytes to follow. If the attacker can corrupt this and specify a number of bytes significantly larger than the amount of buffer space available, he could overrun a buffer.

1. http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

APIs

Function Name	Comments
fread	
pread	
read	
fstat	

Method of Attack

Overflows can occur as the result of an outright attack. For example, in some situations, an attacker can corrupt data for a buffer overflow during a file read. If the number of characters to be read is larger than the buffer space allocated, a buffer overflow will occur.

Other scenarios occur when, for example, the read() function is embedded within a loop. Very often care is not taken to ensure that the maximum number of bytes in the target is not overrun while in the middle of such a loop.

Exception Criteria

Solutions

Solution Applicability	Solution Description	Solution Efficacy
Generally applicable	Perform explicit checks to ensure that the buffer is not exceeded. Ensure that the buffer is null terminated.	

Signature Details

ssize_t read (int fildes , void *buf, size_t nbyte)

Examples of Incorrect Code

```
/* 1) signedness - DO NOT DO THIS. */
char *buf;
int i, len;

read(fd, &len, sizeof(len));

/* OOPS! We forgot to check for < 0 */
if (len > 8000) { error("too large length"); return; }

buf = malloc(len);
read(fd, buf, len); /* len casted to unsigned and overflows */
```

```
/* An example of an ERROR for some 64-bit architectures,
if "unsigned int" is 32 bits and "size_t" is 64 bits: */
```

	`` void *mymalloc(unsigned int size) { return malloc(size); } char *buf; size_t len; read(fd, &len, sizeof(len)); /* we forgot to check the maximum length */ /* 64-bit size_t gets truncated to 32-bit unsigned int */ buf = mymalloc(len); read(fd, buf, len); ``` ``` /* 3) integer overflow */ char *buf; size_t len; read(fd, &len, sizeof(len)); /* forgot to check buffer length */ buf = malloc(len+1); /* +1 can overflow to malloc(0) */ read(fd, buf, len); buf[len] = '\0'; ```
Examples of Corrected Code	``` /* This at least ensures that the buffer is terminated correctly */ read(0, buf, sizeof(buf)-1); buf[sizeof(buf)-1] = '\0'; ```
Source References	• ITS4 Source Code Vulnerability Scanning Tool [2] • read() man page • http://howtos.linux.com/howtos/Secure-Programs-HOWTO/dangers-c.shtml
Recommended Resource	
Discriminant Set	**Operating System**: • Windows **Languages**: • C • C++

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com